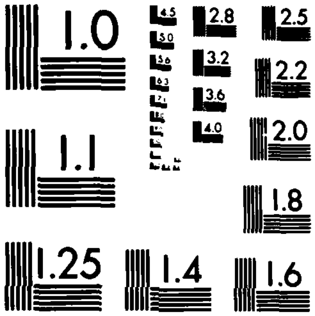
AD-A145 131 ASSESSING THE RELIABILITY OF COMPUTER SOFTWARE AND 1/1
COMPUTING NETWORKS: AN..(U) CALIFORNIA UNIV BERKELEY
OPERATIONS RESEARCH CENTER R E BARLOW ET AL. JUL 84
UNCLASSIFIED ORC-84-7 AFOSR-TR-84-0821 N00014-77-C-0263 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

12

OPERATIONS RESEARCH CENTER

AFOSR-TR-84-0821

AD-A145 131

ASSESSING THE RELIABILITY OF COMPUTER SOFTWARE AND COMPUTING NETWORKS: AN OPPORTUNITY FOR PARTNERSHIP WITH COMPUTER SCIENTISTS

by

Richard E. Barlow[**] and Nozer D. Singpurwalla[**]

ORC 84-7 July 1984

UNIVERSITY OF CALIFORNIA

BERKELEY

DTIC FILE COPY

84 08 29 020

ASSESSING THE RELIABILITY OF COMPUTER SOFTWARE AND COMPUTING NETWORKS: AN OPPORTUNITY FOR PARTNERSHIP WITH COMPUTER SCIENTISTS

by

Richard E. Barlow[†] and Nozer D. Singpurwalla[††]

ORC 84-7 July 1984

[†] Department of Industrial Engineering and Operations Research, University of California, Berkeley, California.

[††] Department of Operations Research, The George Washington University, Washington, D.C.

This research has been partially supported by the Air Force Office of Scientific Research (AFSC), USAF, under Grant AFOSR-81-0122 with the University of California, the Office of Naval Research under Contract N00014-77-C-0263, and the Naval Air Systems Command under Contract N00019-837-C-0358 with The George Washington University. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ORC 84-7	2. GOVT ACCESSION NO. A145131	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* ASSESSING THE RELIABILITY OF COMPUTER SOFTWARE AND COMPUTING NETWORKS: AN OPPORTUNITY FOR PARTNERSHIP WITH COMPUTER SCIENTISTS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard E. Barlow and Nozer D. Singpurwalla		8. CONTRACT OR GRANT NUMBER(s) AFOSR-81-0122
9. PERFORMING ORGANIZATION NAME AND ADDRESS Operations Research Center University of California Berkeley, California 94720		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2304/A5
11. CONTROLLING OFFICE NAME AND ADDRESS United States Air Force Air Force Office of Scientific Research Bolling Air Force Base, D.C. 20332		12. REPORT DATE July 1984
		13. NUMBER OF PAGES 10
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* Computer Reliability Software Reliability Network Reliability		
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)* (SEE ABSTRACT)		

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

For
DTIC COPY INSPECTED 2
A1

ASSESSING THE RELIABILITY OF COMPUTER SOFTWARE AND COMPUTING NETWORKS: AN OPPORTUNITY FOR PARTNERSHIP WITH COMPUTER SCIENTISTS

Richard E. Barlow[*] and Nozer D. Singpurwalla[†]

University of California, Berkeley, California;
The George Washington University, Washington, D.C.

The authors

In this paper, we highlight three areas which are of importance in computer science, and in which statisticians can make valuable contributions. We outline these areas, survey the developments, and point out some of the open problems. The research [illegible] are

1. INTRODUCTION

An aspect of computer science which appears to be largely overlooked by statisticians is that which pertains to assessing the *quality of a piece of software*, or the *trustworthiness of computer hardware and computing networks*. Much of the work in this research area is probabilistic in nature, but is undertaken by engineers and is published in the engineering literature. However, there do remain many unsolved problems which statisticians could help solve and whose solution could influence the state of the art of computer science, especially as it relates to the design of hardware, the configuration of a network of computers, and policies of software development. Thus, these problems present a good opportunity for statisticians to develop a genuine partnership with computer scientists. This is in contrast with the present situation wherein statisticians are using the fruits of computer technology but are not working on the statistical problems presented by computer technology. It appears that much of the previous activity classified as the interface between computer science and statistics, has focused on the above aspect. In this paper we hope to whet the statisticians' appetite by highlighting some of the issues that would constitute an additional dimension to the interface between computer science and statistics. We point out here three research areas, software reliability, the reliability of fault tolerant computers, and the reliability of computer networks.

2. SOFTWARE RELIABILITY

As a result of the rapid technological advances in microelectronics and microprocessors, problems of reliability have changed from those involving hardware to those involving software. According to recent estimates, software costs amount to about 60-80% of total system costs, and software failures account for some of the major difficulties in the operation of commercial and military systems. Software has become a critical component for the navigation and control of aerospace systems, for the successful use of medical care systems, for systems of interest to national defense, and for communication systems. The failure of such systems typically results in some serious economic and strategic consequences, and it is now a well noted fact that software failures are often the primary cause of system failure. To cite an example, it was the failure of software that caused the postponing of a recent mission of the Space Shuttle.

Considerations of the above type have given birth to the subject of *software engineering* of which the assessment of software quality is an important element. Large corporations like A.T.&T.'s Bell Laboratories, the International Business Machines Corporation, British Aerospace, and Government Laboratories such as NASA at Goddard and Langley, have instituted major efforts directed towards the study of software quality and software reliability. Software reliability is a measure of software quality; it is the probability of failure free operation of the software, in a specified environment, for a specified period of time. The quantification and the measurement of software reliability, an assessment of the changes in software reliability over time (reliability growth), the analysis of software failure data, and the decision of whether to continue or to stop testing software, are issues which the statistician is well qualified to address.

2.1 Models for Describing Software Reliability

To facilitate answering some of the above issues, several probabilistic models which describe the stochastic behavior of software running times have been proposed in the literature. A good model can aid program developers estimate the time (and effort) required to achieve a desired level of reliability. Features which distinguish software reliability models from hardware reliability models stem from the fact that software, unlike hardware, does not age with time, and that software failures are due to human errors of design and logic. In contrast to physical failures, the problems of man-made faults have not been suddenly alleviated by a technological breakthrough similar to the invention of semiconductor and ferrite core components. Consequently, once a software error is detected and diagnosed, it is corrected forever, with the net result that after a long span of time the software tends to be nearly perfect. Furthermore, whereas hardware can be replicated (made redundant) to enhance reliability, software replication just replicates the bugs. In what follows, we describe a simple "shock model" for software failures which reflects some of the above features. The model, when generalized, encompasses many of the other models which have been proposed.

Let N^* be the (known) number of distinct "input types" to the software; N^* is assumed large relative to N, the number of input types which result in software failures. N is assumed unknown. Let us assume that the processing of an input is instantaneous, and that once failure occurs, errors in the logic or the coding of the program are corrected, with the net result that N is reduced by 1. If inputs to the software occur according to the postulates of a Poisson process with intensity ω then given N^* and N, the probability of no failure in time $[0,t)$

$$\sum_{j=0}^{\infty} \frac{e^{-\omega t}(\omega t)^j}{j!}\left(\frac{N^*-N}{N^*}\right)^j = e^{-\frac{N}{N^*}\omega t}. \qquad (2.1)$$

This is the familiar shock model for failures.

Let T_i denote the time between (i-1)-st and the i-th failure, $i \le N$. Given N^* and N, the survival function of T_i is clearly

$$\bar{F}_i(t \mid N,N^*,\omega) = \exp(-\omega(N-i+1)t/N^*),\ t \ge 0. \qquad (2.2)$$

The above model is due to Langberg and Singpurwalla (1984) who also show that the several commonly used models for describing software failures are special versions of (2.2). For example, if we let $\Lambda = \omega/N^*$, then the model of Jelinski and Moranda (1972) would result.

From (2.2) above we note that the failure rate of T_i is constant in time, and that the failure rate of T_i is greater than the failure rate of T_{i+1}, $i = 1,2,\ldots,N$; this latter property describes reliability growth.

Since N and Λ are unknown, we should assess meaningful prior distributions for them. Suppose that the prior distribution for N is Poisson with mean θ, and let us assume, for the time being, that $\Lambda = \lambda$ is known. Then Langberg and Singpurwalla also show that M(t), the cumulative number of failures in $[0,t)$ can be described by a nonhomogeneous Poisson process with $EM(t) = \theta(1-e^{-\lambda t})$. This is precisely the model considered by Goel and Okumoto (1979). If on the other hand N = n were assumed known and the prior distribution of Λ were a gamma distribution, then the model considered by Littlewood and Verrall (1973) would result. The choice of other prior distributions for N and Λ, and joint prior distribution for N and Λ would result in other probability models for describing software failures. A generalization of (2.1) resulting from the assumption that the intensity ω is a function of time would lead to still other models for software reliability. For example, the periodic clustering of software failures observed by Crow and Singpurwalla (1984) suggest the possibility of assuming a cyclical intensity function for the underlying Poisson process. Other models for describing the discovery and the correction of errors, the latter being possibly imperfect, are due to Claudio and Kaspersen (1981) and Kremer (1983). These involve the use of birth and death processes.

Despite the above efforts, the development of new models for more realistically describing the stochastic nature of software failures is an actively ongoing effort in which statisticians can make valuable contributions.

2.2 Statistical Inference, Predictive Distributions and Stopping Rules

The fact that the maximum likelihood estimator of the parameter N in the model (2.2) can be highly misleading and often nonsensical, has been noted by Forman and Singpurwalla (1977). That such a thing can happen for any model describing reliability growth has been pointed out by Meinhold and Singpurwalla (1983) who advocate a Bayesian approach. Thus, for example, for any $k \le N$, and any $\underset{\sim}{t}^{(k)} = (t_1,\ldots,t_k)$, where t_i is a realization of T_i, if we assume that the prior distribution of N is a Poisson with a parameter θ, and the prior of Λ is a gamma with scale parameter μ and shape parameter α, independent of the distribution of N, then for any $q \ge N$, the posterior probability for N = q and Λ = T is

$$P\{N = q,\ \Lambda = \lambda \mid \underset{\sim}{t}^{(k)}\}$$

$$= \frac{\lambda^{\alpha+k-1} e^{-\lambda\left[\mu+\sum_{j=1}^{k}(q-j+1)t_j\right]}}{\Gamma(\alpha+k)}$$

$$\times \frac{q!}{(q-k)!}\,\frac{P\{N=q\}}{\sum_{r=k}^{\infty}\frac{r!}{(r-k)!}\left[\mu+\sum_{j=1}^{k}(r-j+1)t_j\right]^{-(\alpha+k)} P\{N=r\}}$$

where $P\{N=j\} = e^{-\theta}\,\theta^j/j!$.

The above result enables us to make inferences about the parameters N and Λ of the Jelinski-Moranda model, from which inferences about T_{k+1} (i.e., its predictive distribution) can be ma le.

In practice, it may sometimes happen that the process of testing and correcting a piece of software does not necessarily lead to its improvement. New errors may be introduced into the program during the act of correcting a previously discovered error. Thus one is often interested in ascertaining if the reliability of the software is indeed improving over time; that is, testing for reliability growth. Knowledge of this kind may influence existing policies of coding and programming, such as team programming, modularized programming, etc. Kyparisis and Singpurwalla (1984) propose a nonhomogeneous Poisson process model with a Weibull intensity function, for testing for reliability growth, and for obtaining the predictive distribution of the next time to failure. A Bayes Empirical Bayes approach involving the use of a simple power law model for reliability growth has been proposed by Horigome, Soyer and Singpurwalla (1984). Whereas such approaches do produce meaningful results when applied to certain sets of actual failure data, they are lacking when applied to some other sets of data. Thus the need for some additional statistical technology for assessing software reliability growth does exist, and here again statisticians can make further contributions.

A final issue pertaining to software quality that we mention here, and one that is unresolved, is the one which addresses the question of when to stop testing the software and release it for its intended use. For a lack of a better term, we refer to this as the *stopping rule problem*, and invite the attention of statisticians to consider this issue, because it is only they who are qualified to come to serious grips with it. Some preliminary work by Forman and Singpurwalla (1979) has been undertaken, bur is less than satisfactory because it lacks a proper balancing between the risk of software failure and the added "information" gained by additional testing of the software.

3. THE RELIABILITY OF FAULT-TOLERANT COMPUTERS

Whereas software reliability problems have attracted the attention of some statisticians, input from statisticians addresing issues pertaining to the reliability of fault-tolerant computers is virtually non-existent. A fault-tolerant computer is a complex system whose key features are the automatic detection, diagnosis, and correction of errors (faults). The system is therefore necessarily dynamic, and assessing its reliability is a non-trivial matter, both from a conceptual as well as a computational point of view. Valuable contributions can be made by statisticians for modelling and evaluating the operation of such important and fast evolving systems. In what follows, we shall introduce the notion of fault-tolerance and its implementation in designing computer systems of the future. In the sequel, we also hope to highlight some of the issues of interest (to statisticians), that such systems pose.

3.1 Fault-Tolerance and Fault-Tolerant Computers

Fault-tolerance is an attribute of a digital system that keeps the logic machine doing its specified tasks when its host, the physical system, suffers various kinds of failures of its components [Avizienis (1978)]. A more general concept of fault-tolerance also includes human mistakes committed during software and hardware implementation and during man/machine interaction. In the quest for reliability, the only alternative to fault-tolerance is *fault-avoidance* or *fault intolerance*. This requires that the physical and the software components of a system, and their assembly techniques be as nearly perfect (failure-free) as possible. The combination of fault-avoidance and manual maintenance is currently the methodology employed for reliability assurance with most computer systems. Noteworthy of these are those in which failure can place human life in danger, a failure which causes heavy economic penalties, and the use of systems which operate in environments that do not allow access for manual maintenance such as spacecraft and unmanned underwater stations. An important example involving a substantial use of fault-tolerant computers is NASA's efforts in designing avionics (airplane) computers of the future.

Avionics computers of the 1990's will be self repairing, and are so designed that they remain operational despite multiple hardware and software failures. The pilot is to be excluded from the flight-stabilizing loop, since no human intervention could be precise enough to handle the advanced airframes of the 1990's. The ultra-reliable, fault tolerant avionic computers will function unattended, despite hardware and software failures, for at least a 10-hour flight. This super-reliability would be gained through redundant hardware and software; faults that occur would be counteracted automatically by hardware and software algorithms. The Federal Aviation Administration (FAA) has no exact reliability requirements for aircraft systems, but the figure that NASA uses call for 10^{-9} as the expected number of failures in a 10-hour flight [Bernhard (1980)]. Conventional avionics computers, by contrast have 10^{-2} to 10^{-3} as the expected number of failures for a 10-hour flight. It is clear from the above, that conventional bench and field tests would take years to perform and thus cannot be used to certify reliability. The figure 10^{-9} is almost impossible to estimate with adequate confidence by conventional tests and methods. The FAA however, has adopted a reliability certification program that emphasizes probabilistic models and simulations of the system; techniques with which the statistician is well acquainted.

Other notions central to fault-tolerance are *error detection, error recovery*, and *fault recovery*. Error detection is the process of discovering that the system is in an erroneous state, and error recovery is undertaken to restore the system to a satisfactory state from an erroneous one, and thus avoid the potential of a system failure. There are two strategies for error recovery, the *backward error recovery* and the *forward error recovery*. With the former, one undoes the work which was previously done to return the system to a previous acceptable state. With foreward error recovery, we either ignore some or all of the work at hand and carry one with only that which we believe to be correct, or send compensating information which corrects erroneous behavior. If we use the term fault to denote the mechanistic cause of an error, then fault recovery consists of removing a faulty component from service and replacing it.

3.2 Implementing Fault-Tolerance

The fundamental principle by which fault-tolerance is achieved in practice is via the provision of redundancy (spares) of resources which may fail, be damaged, or cease to be accessible. Examples of these are processors, memory, and data. Data is particularly important as it is useless having spare processing power and memory, if a component failure causes the data to be irretrievably lost. The notion of *error recovery* deals with ways of ensuring that redundant copies of data are available.

Redundancy is of two types, *static* (or parallel) and *dynamic* (or stand-by). With the latter, the spare resources are not in use when the system is operating normally and are brought into use after component failure. With the former, the spare resources are in use when the system is running normally, and they mirror the operation of the components they are to replace. Dynamic redundancy is usually more flexible, but the length of time taken to recover from a component failure will normally be greater than for static redundancy. Under dynamic redundancy, the reallocation of a spare resource to a failed one, is usually performed under program control. This may be a relatively slow process and may cause the system to fail because it does not meet some deadline. To avoid this, a fault tolerant system must have *time redundancy*. That is, there must be sufficient spare (processing) capacity in the system, for reallocation to take place, after a component failure. Time redundancy is also useful in achieving fault tolerance when one is dealing with what are known as transient physical faults - these are faults whose manifestation does not last longer than a certain maximum time. A classic example of the use of time redundancy in dealing with transient faults, is a communication system where packets corrupted by noise are retransmitted.

In the context of computers, redundancy can be achieved via *multiple processors* or *multiple computers*. In the former, we have many processors communicating via a shared main store. In principle, the processors can readily check each other, share and communicate the data with each other, and in the event of a failure, one processor can take over the work of the other. The processors in a multi-processor system are physically very close making them susceptible to common mode failures, or inducing statistical dependencies among their life lengths.

In multiple computer systems, independent computers communicate via some communication medium and constitute a network of computers. The main advantage of such a system emanates from the fact that physical separation of the computers increases the capacity of the system to withstand damage at a particular location. The difficulties and problems of computing the reliability of such systems is discussed in Section 4.

3.3 The Use of Redundancy for Achieving Fault-Tolerance

A simple (non-redundant) circuit has no protection against component failures - the failure of a component may lead to the failure of the circuit. If there is no mechanism for monitoring the circuit's behavior, then the failure of the circuit may not be immediately apparent. Thus a simplex circuit is not fault-tolerant.

A duplex system consists of two identical circuits in static redundancy, and the output of the two circuits are compared. The failure of any component in either circuit will cause a discrepancy between the outputs of the two circuits. When this happens the system is said to have failed, because we are not able to determine which of the two circuits have failed. We have failure detection but not fault tolerance. The reliability of the duplex system is less than the reliability of the simplex.

The addition of a further redundant circuit yields the classical Triple Modular Redundant (TMR) circuit. A failure in any circuit would lead to a discrepancy between its output and that of the other two. By using the majority voting rule, the comparing device selects the correct output and sends this on to the next stage. Thus, we have achieved fault and error detection. The TMR is one of the simplest fault tolerant systems. After one failure, the TMR reduces to a duplex system. We can verify that the TMR system is the most reliable in the short term. The comparison device is generally simpler than the circuits, and so it is reasonable to assume that it is not likely to fail.

We can continue to increase the level of redundancy provided, and produce a quadruple redundant circuit. We can also replicate the comparison (voter) devices to protect against failure of the voting system. TMR systems with dynamic replacement of failed components to maintain fault recovery have been built. A disadvantage of TMR systems, is that the simultaneous failure of two or more circuits due to a common mode (or a shock), when not detected could lead to an erroneous output. The failed circuits would constitute a majority vote and send an erroneous output to the next stage.

3.4 Reliability Analysis of Fault-Tolerant Computers

It should be clear from our discussion of the triple modular redundant circuit that a collection of several such circuits in a large computer would quickly cascade into several stages posing difficult computational problems for the reliability analysis of the computer. The problems will be further aggravated if we have quadruple redundant circuits (which are more reliable than the TMR circuits) or if we replicate the comparison devices to ensure further reliability. As of now, no satisfactory method for estimating the reliability of such systems exists. For NASA's avionics computer, one has to deal with 10^6 states if the problem were to be approached via a Markov chain model. NASA engineers use the *Computer Aided Reliability Estimation* III (CARE III) procedure for undertaking such analyses. An overview of CARE III and a chronology of its development is given by Bavuso (1984). A tutorial on CARE III, which is more detailed than Bavuso's paper, is given by Trivedi and Geist (1981). The potential for refinement of the CARE III approach exists, including the possibility of looking at the problem from an entirely different point of view. We draw the attention of statisticians to this problem.

4. A SURVEY OF EFFICIENT NETWORK RELIABILITY COMPUTATIONAL METHODS

A classic problem involving computer networks is to calculate the probability that certain distinguished nodes (computers) in a network are connected. Arcs and/or nodes in the network may fail, thus disconnecting some computer pairs. The difficulty of the problem arises partly from the complexity of the network. These problems are known to be, in the worst case, NP-hard. Hence, efficient (i.e., polynomial time algorithms) are only likely for special structures. However, it has been conjectured that, in particular, these problems can be solved efficiently for planar networks. Further complications arise when arc and/or computer success probabilities are not known but their uncertainty is assessed in the form of a probability density, perhaps based on arc and/or computer failure data. Although the problem can, in principle, be solved using standard Bayesian procedures, it can be solved in practice only with the aid of efficient computational algorithms. In the next sections we discuss the problem in more depth and describe the best known methods available at this date.

4.1 The Statistical Problem

Given a network such as the ARPA computer network (circa 1974) in Figure 1, we wish to calculate the probability that the distinguished nodes (dark circles) can communicate with each other. (In general, let K be the set of distinguished nodes.) The distinguished nodes will be connected if there is at least one K-tree, all of whose arcs work. (A K-tree of a graph, G, with respect to K, is any minimal graph which connects all the distinguished nodes in K.) A K-tree is the analogue of a minimal path for a 2-terminal network problem.

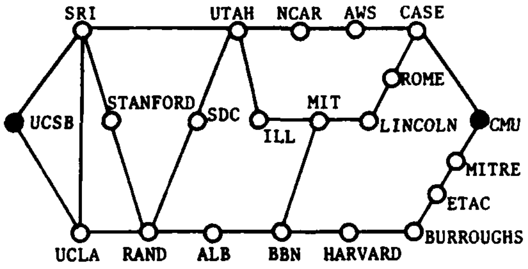


Figure 1: ARPA Computer Network ~ 1974

For simplicity we will assume henceforth that only arcs can fail. There are at least three possible complications:

1. In our judgement, arcs may not fail independently of one another given arc reliability parameters $p_1, p_2, \ldots, p_n$;
2. In our judgement, some arcs may have identical reliability parameters so that they are a priori dependent relative to the joint prior density for parameters;
3. The network may be topologically very complex.

We will use the following notation. Let $X_i = 1$ if arc i works and 0 otherwise. Let $\phi(X_1, X_2, \ldots, X_n) = 1$ if the distinguished nodes are connected and 0 otherwise. In principle, ϕ can be represented in sums of products form and our problem can be solved using this representation. In practice, this is not feasible for large networks.

A. Satyanarayana (1982) obtained the most efficient algorithm for finding $\phi(X_1, X_2, \ldots, X_n)$ within the class of methods based on the inclusion-exclusion principle. However, this is an exponential time algorithm in general. The so-called topological formula obtained is of the form

$$\phi(X_1, X_2, \ldots, X_n) = \sum_i (-1)^{b_i - v_i + 1} \prod_{j \varepsilon G_{a,i}} X_j$$

where $G_{a,i}$ is the i-th acyclic K-subgraph with

b_i arcs and v_i vertices or nodes. A K-subgraph is a connected directed graph for a given K with every arc in some K-tree with respect to K. If the original network is undirected (which is true for the network of Figure 1) then it must be directed by replacing each arc by two arcs in anti-parallel. However, arcs in such anti-parallel pairs can be assumed to fail independently of each other but have the same reliability parameter. Independence may be assumed since only one member of each pair will appear in any acyclic K-subgraph. A Pascal computer program for solving the 2-terminal problem is available from the authors.

Let the network reliability parameter be

$$h = P[\phi(x_1, x_2, \ldots, x_n) = 1 \mid \pi(\cdot)]$$

where $\pi(\cdot)$ is the joint probability density for arc reliability parameters $(p_1, p_2, \ldots, p_n) \overset{\text{DEF}}{=} \underline{p}$. Since h will have a probability density induced by $\pi(p_1, \ldots, p_n)$, the unconditional network reliability will be $E[h \mid \pi(\cdot)]$. If arcs are judged to fail independently of each other then

$$Eh = E_{\underline{p}} h(\underline{p}) = E_{\underline{p}} \left\{ \sum_i (-1)^{b_i - v_i + 1} \prod_{j \epsilon G_{a,i}} p_j \right\}.$$

If, furthermore, arc reliability parameters are judged independent, then

$$Eh = E_{\underline{p}} h(\underline{p}) = \sum_i (-1)^{b_i - v_i + 1} \prod_{j \epsilon G_{a,i}} E(p_j).$$

4.2 The Variance of the Network Reliability Parameter h

Assume $\overset{n}{\underset{i=1}{\perp}} x_i \mid \underline{p}$ and $\overset{n}{\underset{i=1}{\perp}} p_i \mid \pi(\cdot)$ where $\perp$ indicates independence. Given $\underline{p}$, $h(\underline{p})$ is the network reliability. Since $\underline{p}$ has joint density $\pi(\cdot)$, $\mathrm{Var}_{\underline{p}}[h(\underline{p})]$ may be of interest as well as

$$E_{\underline{p}} h(\underline{p}) = h(E\underline{p}).$$

Var[h] can be approximated by calculating expressions of the form $h(\underline{1}_I, \underline{0}_J, E\underline{p})$ where $(\underline{1}_I, \underline{0}_J, E\underline{p})$ is the $E\underline{p}$ vector with coordinates replaced by 1 whose indices are in I and coordinates whose indices are in J are replaced by 0.

Using Taylor's formula for several variables, we have

$$h(\underline{p}) - h(E\underline{p})$$

$$= \sum_{k=1}^{m} \frac{1}{k!} d^k h(E\underline{p}; \underline{p} - E\underline{p}) + \frac{1}{m!} d^m h(\underline{z}; \underline{p} - E\underline{p})$$

where $d'h(E\underline{p}; \underline{p} - E\underline{p}) = \sum_{i=1}^{n} [h(1_i, E\underline{p}) - h(0_i, E\underline{p})] \times (p_i - Ep_i)$, etc. and $\underline{z}$ is on the line segment joining $\underline{p}$ and $E\underline{p}$. Therefore, to the first order ($m = 1$),

$$\mathrm{Var}[h(\underline{p})] \doteq \sum_{i=1}^{n} [h(1_i, E\underline{p}) - h(0_i, E\underline{p})]^2 \, \mathrm{Var}(p_i).$$

Better approximations can be obtained by including more terms. If efficient (i.e., linear or polynomial time) methods can be found for computing $h(\underline{1}_I, 0_J, E\underline{p})$, this will be a practical method for approximating $\mathrm{Var}_{\underline{p}}[h(\underline{p})]$.

4.3 Efficient Calculation of $h(E\underline{p})$

There have been recent breakthroughs in the development of efficient algorithms for computing $h(E\underline{p})$ when $\overset{n}{\underset{i=1}{\perp}} x_i \mid \underline{p}$ and $\overset{n}{\underset{i=1}{\perp}} p_i \mid \pi(\cdot)$. These apply to planar networks of special structure, both directed and undirected.

Consider the undirected graph in Figure 2. Efficient methods now exist to calculate the probability that all nodes in such a graph can communicate (cf. Politof and Satyanarayana (1984)). The graph may be represented by an adjacency matrix or an adjacency structure [the list of all successors (neighbors) of each vertex].

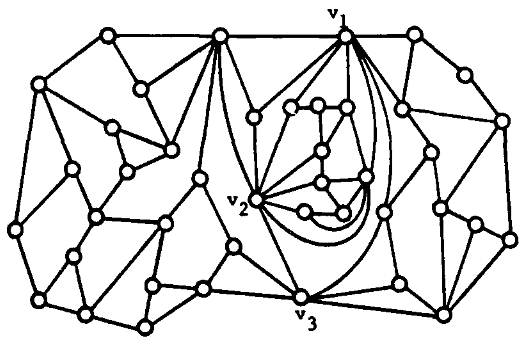


Figure 2: Example Graph

The approach used is roughly as follows:

(i) Decompose the graph into its triconnected components (a graph is triconnected if it has no separation pairs of vertices.)

(ii) Find a planar embedding for each triconnected component.

(iii) If (ii) is successful, use Δ-Y and polygon-to-chain probability reductions to determine the reliability of each component.

(iv) Calculate the reliability of the network using the calculated reliabilities of each of its components.

The previous approach is currently being implemented for the all terminal network reliability problem. The associated algorithm has running time of order $|V|^2$ where $|V|$ is the number of vertices or nodes.

Linear running time computer programs are now available for both directed and undirected networks when the graph is basically series-parallel. Figure 3 shows a series-parallel graph. A graph is basically series-parallel if it can be reduced to a tree by series and parallel *replacements* (as contrasted with series and parallel *probability reductions*).

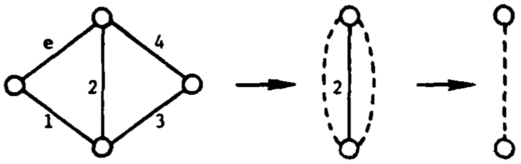


Figure 3: Series and Parallel Replacements

Figure 4 shows a graph which is *not* basically series-parallel.

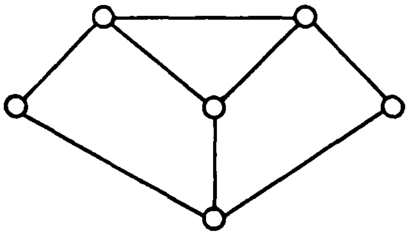


Figure 4: Example Network which is *not* Basically Series-Parallel

4.3 Available Computer Programs

No single computer program is available for efficiently computing the reliability of arbitrary planar networks. However, the following list covers special structures of practical interest.

1. *General Directed Graphs* - (2 terminal).
TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM
Pascal (Berkeley Pascal PI - Version 2.0)
DON, SIYI March 1981.

2. *Undirected Basically Series Parallel Graphs.*
POLYCHAIN - (K terminal)
Fortran IV (standard)
Mauricio G. C. Resende
October 1983.

3. *Directed Basically Series Parallel Graphs.*
NETWORK
Pascal (Berkeley Pascal PI - Version 3.0)
Jack Shaio
December 1983
2-terminal.

FOOTNOTES

*This research was supported by the Air Force Office of Scientific Research (AFSC), USAF, under Grant AFOSR-81-0122 with the University of California. Reproduction in whole or in part is permitted for any purpose of the United States Government.

†Supported by the Office of Naval Research under Contract N00014-77-C-0263 Project NR-042-372, and the Naval Air Systems Command under Contract N00019-83-C-0358 with the George Washington University.

REFERENCES

[1] Agrawal, A. and Barlow, R. E., A Survey of Network Reliability, ORC 83-5, Operations Research Center, University of California, Berkeley (July 1983); to appear in Operations Research.

[2] Avizienis, A., Fault-tolerance: The survival attribute of digital systems, Proceedings of the IEEE (1978) 1109-1125.

[3] Bavuso, S. J., A User's Overview of CARE III, NASA Langley Research Center, Hampton, VA (1984).

[4] Bernhard, R., The 'no-downtime' computer, IEEE Spectrum (September 1980) 33-37.

[5] Claudio, L. F. and Kaspersen, D. L., Example applications of software reliability estimation, Proceedings of the 1981 Army Numerical Analysis and Computers Conference (1981) 41-58.

[6] Crow, L. H. and Singpurwalla, N. D., An empirically developed fourier series model for describing software failures, to appear in IEEE Trans. Reliability.

[7] Forman, E. H. and Singpurwalla, N. D., An empirical stopping rule for debugging and testing computer software, J. Amer. Statis. Assoc. 72 (1977) 750-757.

[8] Forman, E. H. and Singpurwalla, N. D., Optimal time intervals for testing hypothesis on computer software errors, IEEE Trans. Reliability R-28 (1979) 250-253.

[9] Goel, A. L. and Okumoto, K., Time dependent error detection rate model for software reliability and other performance measures, IEEE Trans. Reliability R-28 (1979) 206-211.

[10] Horigome, M., Singpurwalla, N. D. and Soyer, R., A Bayes empirical Bayes approach for (software) reliability growth, Proceedings of the 16th Computer Science and Statistics Symposium on the Interface (L. Billard, Ed.) (Atlanta, Georgia, March 14-16, 1984), North Holland, to appear.

[11] Jelinski, Z. and Moranda, P. B., Software reliability research, in Statistical Computer Performance Evaluation (W. Freiberger, Ed.) (Academic Press, New York, 1972).

[12] Kremer, W., Birth-death and bug counting, IEEE Trans. Reliability R-32 (1983) 37-46.

[13] Kyparisis, J. and Singpurwalla, N. D., Bayesian inference for the Weibull process with applications to assessing software reliability growth and predicting software failures, Proceedings of the 16th Computer Science and Statistics Symposium on the Interface (L. Billard, Ed.) (Atlanta, Georgia, March 14-16, 1984), North Holland, to appear.

[14] Langberg, N. and Singpurwalla, N. D., Unification of some software reliability models, SIAM J. Sci. Statis. Comp. (1982), to appear.

[15] Littlewood, B. and Verral, J. L., A Bayesian reliability growth model for computer software, Record IEEE Symp. Comp. Software Reliability (1973) 70-77.

[16] Meinhold, R. J. and Singpurwalla, N. D., Bayesian analysis of a commonly used model for describing software failures, The Statistician 32 (1983) 168-173.

[17] Politof, T. and Satyanarayana, A., An $O(|V|^2)$ Algorithm for a Class of Planar Graphs to Compute the Probability that the Graph is Connected, Operations Research Center, University of California, Berkeley (1984).

[18] Resende, M., A Computer Program for Reliability Evaluation of Large-Scale Undirected Networks Via Polygon-to-Chain Reductions, ORC 83-10, Operations Research Center, University of California, Berkeley (1983).

[19] Satyanarayana, A., A unified formula for the analysis of some network reliability problems, IEEE Trans. Reliability R-31 (April 1982) 23-32.

[20] Satyanarayana, A. and Agrawal, A., Source to K Terminal Reliability Analysis of Rooted Communication Networks, ORC 83-2, Operations Research Center, University of California, Berkeley (February 1983); submitted for publication to Networks.

[21] Satyanarayana, A. and Wood, R. K., Polygon-to-Chain Reductions and Network Reliability, ORC 82-4, Operations Research Center, University of California, Berkeley (March 1982); submitted for publication to SIAM J. Computing.

[22] Trivedi, K. S. and Geist, R. M., A Tutorial on the CARE III Approach to Reliability Modeling, NASA Langley Research Center, Hampton, VA (1981).

END

FILMED

10-84

DTIC